**FUNCTIONS**

1. **Function Declaration**
   * A function declaration is defined using the function keyword.
   * Function declarations are hoisted, making them available throughout the scope, which can be useful in various scenarios.

function calculateRectangleArea(width, height) {

  return width \* height;

}

function calculateCircleArea(radius) {

  return Math.PI \* radius \*\* 2;

}

console.log(calculateRectangleArea(4, 5));

console.log(calculateCircleArea(3));

1. **Function Expression**
   * A function expression involves defining a function as part of an expression, often assigning it to a variable.
   * Useful when you need to conditionally assign a function to a variable or when you want to create functions dynamically.

const powerFunction = function(base) {

    return function(exponent) {

        return Math.pow(base, exponent);

    };

};

const square = powerFunction(2);

console.log(square(3));

const cube = powerFunction(3);

console.log(cube(2));

1. **Arrow Functions**
   * A concise way to write functions introduced in ES6.
   * Great for short, simple functions, especially when writing callback functions or in situations where a concise syntax is preferred.

const squared = numbers.map((num) => num \*\* 2);

console.log(squared);

const evens = numbers.filter((num) => num % 2 === 0);

console.log(evens);

1. **Named Function**
   * A standard function declaration with a name.
   * Named functions are useful when you want a clear, reusable block of code. They are hoisted, meaning they can be called before their declaration in the code.

function factorial(n) {

  if (n === 0 || n === 1) {

    return 1;

  } else {

    return n \* factorial (n - 1);

  }

}

console.log(factorial (5));

1. **Anonymous Function**
   * A function without a name, often assigned to a variable.
   * Useful when you need a function for a short-lived task or when assigning a function to a variable for later use.

let calculator = {

  add: function (x, y) {

    return x + y;

  },

  subtract: function (x, y) {

    return x - y;

  },

  multiply: function (x, y) {

    return x \* y;

  },

  divide: function (x, y) {

    return x / y;

  },

};

console.log(calculator.add(5, 3));

console.log(calculator.subtract(9, 3));

console.log(calculator.multiply(5, 6));

console.log(calculator.divide(10, 3));

1. **IIFE (Immediately Invoked Function Expression)**
   * A function that is executed immediately after it's created.
   * Useful for creating a private scope for variables to avoid polluting the global namespace and for executing code immediately.

const counter = (function () {

  let count = 0;

  return {

    increment: function () {

      count++;

    },

    getCount: function () {

      return count;

    },

  };

})();

counter.increment();

counter.increment();

console.log(counter.getCount()); //2

1. **Callback Function**
   * A function passed as an argument to another function and executed later.
   * Commonly used in asynchronous operations, event handling, and for modularity.

function fetchData(callback) {

  setTimeout(function () {

    const data = "Some data";

    callback(data);

  }, 1000);

}

function processResult(result) {

  console.log("Processing result: " + result);

}

fetchData(processResult);

1. **Recursive Function**
   * A function that calls itself.
   * Commonly used for solving problems that can be broken down into smaller, similar sub-problems.

function flattenArray(arr) {

  return arr.reduce((flat, element) => {

    return flat.concat(

      Array.isArray(element) ? flattenArray(element) : element

    );

  }, []);

}

const nestedArray = [1, [2, [3, 4], 5], 6, [7, 8]];

const flatArray = flattenArray(nestedArray);

console.log(flatArray);

1. **Higher-Order Function**
   * A function that takes one or more functions as arguments or returns a function.
   * Useful for creating abstractions and building more flexible and modular code.

function applyToEach(arr, func) {

  return arr.map(func);

}

const number = [1, 2, 3, 4];

const squares = applyToEach(number, (num) => num \*\* 2);

console.log(squares);

1. **Generator Function**
   * A special type of function that can be paused and resumed.
   * Useful for handling asynchronous operations, iterating over large datasets, and implementing custom iteration behaviour.

function\* range(start, end) {

  for (let i = start; i <= end; i++) {

    yield i;

  }

}

for (let num of range(1, 5)) {

  console.log(num);

}